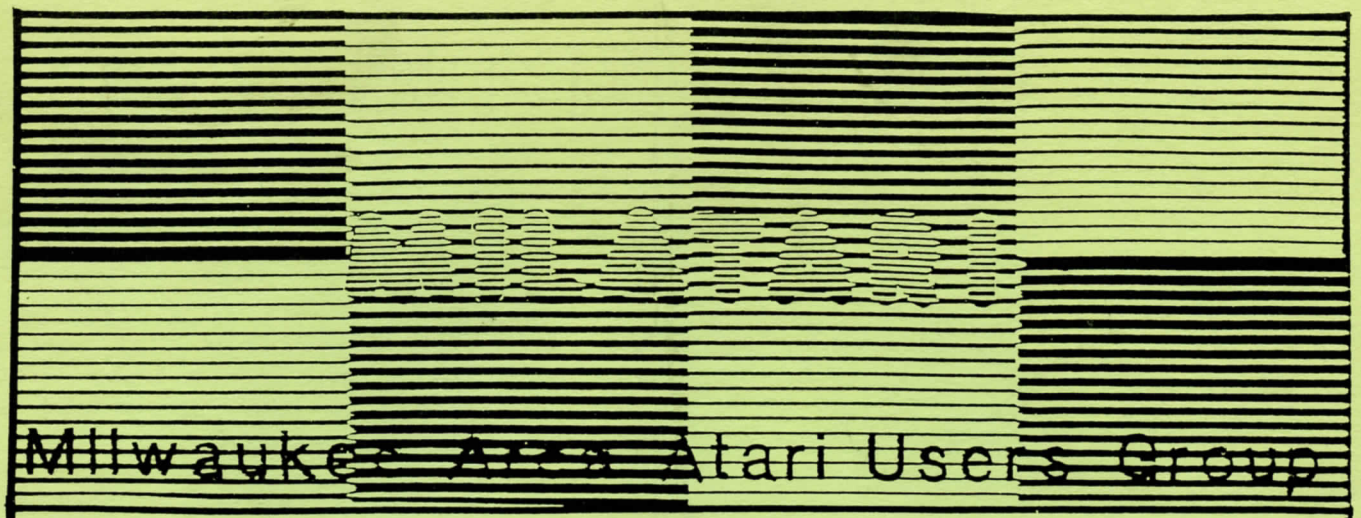


MILATARI NEWSLETTER

Vol.5 Nbr.4 PRICE $1.50 March 1986

Calendar of Events

March 15,1986 Armbruster School
7000 Greenway, Greendale
(Off 68th 1 Block North of Grange)

- 2:15PM - 520ST SIG
- 2:15PM - Basic BASIC Class - Steve Armstrong
- 2:15PM - C Language Class - Mark Manyen
- 3:30PM - SPARTA DOS - Ron Friedel & Gary Haberman

March 20,1986 Board of Directors Meeting 7:00PM
Ground Round Bluemound & Hwy 100

MILATARI BBS 414-781-5710 24 Hours 300 Baud

The Nominating Committee is looking for any members that would like to become an officer of the club and/or your suggestions for club officers. Please bring your suggestions to their attention at or before the meeting

** Professional GEM **
by Tim Oren

Topic: Windows, part II

EXCELSIOR!

In this installment, we continue the exploration of GEM's window manager by finding out how to process the messages received by an application when it has a window defined on the screen.

All references to non-GEM routines in this column refer to the sample C code attached. Please note that this sample code will not contain entire programs. Instead, it consist of small pieces of utility code which you may copy and modify in your own programs.

REDRAWING WINDOWS

One of the most misunderstood parts of GEM is the correct method for drawing within a window. Most requests for redrawing are generated by the GEM system, and arrive as messages (read with evnt_multi) which contain the handle of the window, and the screen rectangle which is "dirty" and needs to be redrawn.

Screen areas may become dirty as a result of windows being closed, sized down, or moved, thus "exposing" an area underneath. The completion of a dialog, or closing of a desk accessory may also free up a screen area which needs to be redrawn. When GEM detects the presence of a dirty rectangle, it checks its list of open windows, and sends the application a redraw message for each of its windows which intersects the dirty area.

CAVEAT EMPTOR

GEM does not "clip" the rectangle which it sends to the application; that is, the rectangle may not lie entirely within the the screen. It is the job of the application to determine in what portion of the rectangle it may safely draw. This is done by examining the "rectangle list" associated with the window.

A rectangle list is maintained by GEM for each active window. It contains the portions of the window's interior which are exposed, i.e., topmost, on the screen and within which the app may draw.

Let's consider an example to make this clear. Suppose an app has opened two windows, and there are no desk accessory windows open. The window which is topmost will always have only one rectangle in its list. If the two are separate on the screen, then the second window will also have one rectangle. If they overlap, then the top window will "break" the rectangle of the bottom one. If the overlap is at a corner, two rectangles will be generated for the bottom window. If the overlap is on a side only, then three rectangles are required to cover the exposed portion of the bottom window. Finally, if the first window is entirely within the second, it requires four rectangles in the list to tile the second window.

Try working out a few rectangle examples with pencil and paper to get the feel of it. You will see that the possible combinations with more than two windows are enormous. This, by the way, is the reason that GEM does not send one message for each rectangle on the list: With multiple windows, the number of messages generated would quickly fill up the application's message queue.

Finally, note that every app MUST use this method, even if it only uses a single window, because there may be desk accessories with their own windows in the system at the same time. If you do not use the rectangle lists, you may overwrite an accessory's window.

INTO THE BITS

First, we should note that the message type for a redraw request is WM_REDRAW, which is

message returned by evnt_multi. The window handle is stored in msg[3]. These locations are the same for all of the message types being discuss. The rectangle which needs to be redrawn is stored in msg[4] through msg[7].

Now let's examine the sample redraw code in more detail. The redraw loop is bracketed with mouse off and mouse on calls. If you forget to do this, the mouse pointer will be over-written if it is within the window and the next movement of the mouse will leave a rectangular blotch on the screen as a piece of the "old" screen is incorrectly restored.

The other necessary step is to set the window update flag. This prevents the menu manager from dropping a menu on top of the screen portion being redrawn. You must release this flag at the end of the redraw, or the you will be unable to use any menus afterwards.

The window rectangles are retrieved using a get-first, get-next scheme which will be familiar if you have used the GEM DOS or PC-DOS wildcard file calls. The end of the rectangle list has been reached when both the width and height returned are zero. Since some part of a window might be off-screen (unless you have clamped its position - see below), the retrieved rectangle is intersected with the desktop's area, and then with the screen area for which a redraw was requested.

Now you have the particular area of the screen in which it is legal to draw. Unless there is only one window in your application, you will have to test the handle in the redraw request to figure out what to put in the rectangle.

Depending on the app, you may be drawing an AES object tree, or executing VDI calls, or some combination of the two. In the AES case, the computed rectangle is used to specify the bounds of the objc_draw. For I work, the rectangle is used to set the clipping area before executing the VDI calls.

A SMALL CONFESSION

At the beginning of this discussion, I deliberately omitted one class of redraws: those initiated by the application itself. In some cases a part of the screen must be redrawn immediately to give feedback to the user following a keystroke, button, or mouse action. In these cases, the application could call do_redraw directly, without waiting for a message.

The only time you can bypass do_redraw, and draw without walking the rectangle list, is when you can be sure that the target window is on top, and that the figure being drawn is entirely contained within it.

In many cases, however, an application initiated redraw happens because of a computed change, for instance, a spreadsheet update, and its timing is not crucial. In this instance, you may wish to have the app send ITSELF a redraw request.

The main advantage of this approach is that the AES is smart enough to see if there is already a redraw request for the same window in the queue, and, if so, to merge the requests by doing a union of their rectangles. In this fashion, the "blinky" appearance of multiple redraws is avoided, without the need to include logic for merging redraws within the program.

A utility routine for sending the "self-redraw" is included in the down-load for this article.

WINDOW CONTROL REQUESTS

An application is notified by the AES, via the message system, when the user manipulates one of the window control points. Remember that you must have specified each control point when the window was created, or will not receive the associated control message.

The most important thing to understand about window control is that the change which the user requested does not take place until the

application forwards it to the AES. While this makes for a little extra work, it gives the program a chance to intervene and validate or modify the request to suit.

A second thing to keep in mind is that not all window updates cause a redraw request to be generated for the window, because the AES attempts to save time with raster moves on the screen.

Now let's look at each window control request in detail. The message code for a window move is WM_MOVED. If you are willing to accept any such request, just do:

```
wind_set(wh, WF_CXYWH, msg[4], msg[5],
msg[6], msg[7]);
```

(Remember that wh, the window handle, is always in msg[3]).

The AES will not request a redraw of the window following this call, unless the window is being moved from a location which is partially "off-screen". Instead, it will do a "blit" (raster copy) of the window and its contents to the new location without intervention by the app.

There are two constraints which you may often wish to apply to the user's move request. The first is to force the new location to lie entirely within the desktop, rather than partially off-screen. You can do this with the rc_constrain utility by executing:

```
rc_constrain(&full, &msg[4]);
```

before making the wind_set call. (Full is assumed to contain the desktop dimensions.)

The second common constraint is to "snap" the x-dimension location of the new location to a word boundary. This operation will speed up GEM's "blit" because no shifting or masking will need to be done when moving the window. To perform this operation, use align() before the wind_set call:

```
msg[4] = align(msg[4], 16);
```

The message code for a window size request is WM_SIZED. Again, if you are willing to accept any request, you can just "turn it around" with the same wind_set call as given for WM_MOVED.

Actually, GEM enforces a couple of constraints on sizing. First, the window may not be sized off screen. Second, there is a minimum window size which is dependent on the window components specified when it was created. This prevents features like scroll arrows from being squeezed into oblivion.

The most common application constraint on sizing is to snap the size to horizontal words (as above) and/or vertical character lines. In the latter case, the vertical dimension of the output font is used with align().

Also, be aware that the size message which you receive specifies the EXTERNAL dimensions of the window. To assure an "even" size for the INTERNAL dimensions, you must make a wind_calc call to compute them, use align() on the computed values, back out the corresponding external dimensions with the reverse wind_calc, and then make the wind_set call with this set of values.

A window resize will only cause a redraw request for the window if the size is being increased in at least one dimension. This is satisfactory for most applications, but if you must "reshuffle" the window after a size-down, you should send yourself a redraw (as described above) after you make the wind set call. This will guarantee that the display is updated correctly. Also note that the sizing or movement of one window may cause redraw requests to be generated for other windows which are uncovered by the change.

The window full request, with code WM FULLED, is actually a toggle. If the window is already at its full size (as specified i the wind_create), then this is a request to shrink to its previous size. If the window is currently small, then the request is to grow to full size.

Since the AES records the current, previous, and maximum window size, you can use wind get calls to determine which situation pertains. The hndl_ full utility in the down-load (modified from Doodle), shows how to do this.

The "zoom box" effects when changing size are optional, and can be removed to speed things up. Again, if the window's size is decreasing, no redraw is generated, so you must send yourself one if necessary. You should not have to perform any constraint or "snap" operations here, since (presumably) the full and previous sizes have had these checks applied to them already.

The WM_CLOSED message is received when the close box is clicked. What action you perform depends on the application. If you want to remove the window, use wind_close as described in the last column. In many applications, however, the close message may indicate that a file is to be saved, or a directory or editing level is to be closed. In these cases, the message is used to trigger this action before or instead of the wind_close. (Folders on the Desktop are an example of this situation.)

The WM_TOPPED message indicates that the AES wants to bring the indicated window to the "top" and make it active. This happens if the user clicks within a window which is not on top, or if the currently topped window is closed by its application or desk accessory. Normally, the application should respond to this message with:

```
wind_set(wh, WF_TOP, 0, 0);
```

and allow the process to complete.

In a few instances, a window may be used in an output only mode, such as a status display, with at least one other window present for input. In this case, a WM TOPPED message for the status window may be gnored. In all other cases, you must handle the WM_TOPPED message even if your application has only one window: Invocation of a desk accessory could always place another window on top. If you fail to do so, subsequent redraws for your window may not be processed correctly.

WINDOW SLIDER MESSAGES

If you specify all of the slider bar parts for your window, you may receive up to five different message types for each of the two sets of sliders. To simplify things a little, I will discuss everything in terms of the vertical (right hand side) sliders. If you are also using the horizontal sliders, the same techniques will work, just use the alternate mnemonics.

The WM_VSLID message indicates that the user has dragged the slider bar within its box, indicating a new relative position within the document. Along with the window handle, this message includes the relative position between 1 and 1000 in msg[4].

Recall from last column's discussion that this interval corresponds to the "freedom of movement" of the slider. If you want to accept the user's request, just make the call:

```
wind_set(wh, WF_VSLIDE, msg[4], 0, 0, 0);
```

(Corresponding horizontal mnemonics are WM HSLID and WF_HSLIDE).

Note that this wind_set call will not cause a redraw message to be sent. You must update the display to reflect the new scrolled position, either by executing a redraw directly, or by sending yourself a message.

If the document within the window has some structure, you may not wish to accept all slider positions. Instead you may want to force the scroll position to the nearest text line (for instance). Using terms defined in the last column, you may convert the slider position to "document units" with:

```
top_wind = msg[4] * (total_doc - seen_doc) /
1000 + top_doc
```

(This will probably require 32-bit arithmetic).

After rounding off or otherwise modifying the request, convert it back to slider units and make the WF_VSLIDE request.

The other four slider requests all share one message code: WM_ARROWED. They are distinguished by sub-codes stored in msg[4]: WA_UPPAGE, WA_DNPAGE, WA_UPLINE, and WA DNLINE. These are produced by clicking above and below the slider, and on the up and down arrows, respectively. (I have no idea why sub-codes were used in this one instance.) The corresponding horizontal slider codes are: WA_LFPAGE, WA_RTPAGE, WA LFLINE, and WA_RTLINE.

What interpretation you give to these requests will depend on the application. In the most common instance, text documents, the customary method is to change the top of window position (top_wind) by one line for a WA_UPLINE or WA_DNLINE, and by seen_doc (the number of lines in the window) for a WA UPPAGE or WA_DNPAGE.

After making the change, compute a new slider position, and make the wind_set call as given above. If the document's length is not an even multiple of "lines" or "pages" you will have to be careful that incrementing or decrementing top_wind does not exceed its range of freedom: top_doc to (top_doc + total_doc - seen_doc).

If you have such an odd size document, you will also have to make a decision on whether to violate the line positioning rule so that the slider may be put at its bottom-most position, or to follow the rule but make it impossible to get the slider to the extreme of its range.

A COMMON BUG

It is easy to forget that user clicks are not the only things that affect slider position. If the window size changes as a result of a WM_SIZED or WM_FULLED message, the app must also update its sliders (if they are present). This is a good reason to keep the top of window information in "document units".

You can just redo the position calculation with the new "seen_doc" value, and call wind set. Also remember that changing the size of the underlying document (adding or deleting a bottom line, for instance) must also cause the sliders to be adjusted.

DEPT. OF DIRTY TRICKS

There are two remaining window calls which are useful to advanced programmers. They require techniques which I have not yet discussed, so you may need to file them for future reference.

The AES maintains a quarter-screen sized buffer which is used to save the area under alerts and menu drop-downs. It is occasionally useful for the application to gain access to this buffer for its own use in saving screen areas with raster copies. To do so, use:

```
wind_ get(0, WF_ SCREEN, &loaddr, &hiaddr,
&lolen, &hilen);
```

Hiaddr and loaddr are the top and bottom 16-bits (respectively) of the 32-bit address of the buffer. Hilen and lolen are the two halves of its length.

Due to a preculiarity of the binding you have to reassemble these pieces before using them. (The actual value of WF_SCREEN is 17; this does not appear in some versions of the GEMDEFS.H file.)

If you use this buffer, you MUST prevent menus from dropping down by using either the BEG_UPDATE or BEG_MCTRL wind_update calls. Failure to do so will result in your data being destroyed. Remember to use the matching wind_ update: END_ UPDATE or END MCTRL, when you are done.

The other useful call enables you to replace the system's desktop definition with a resource of your choosing. The call:

```
wind_set(0, WF_NEWDESK, tree, 0, 0);
```

where tree is the 32-bit address of the object tree, will cause the AES to draw your definition instead of the usual gray or green background. Not only that, it will continue to redraw this tree with no intervention on your part.

Obviously, the new definition must be carefully built to fit the desktop area exactly or garbage will be left around the edges. For the truly sophisticated, a

The FUZZY NOLAN REVIEWS

by Gary Nolan

OH, WHAT A TANGLED WEB WE WEAVE
(When first we practice to .....)

It looks as if 1986 will be the year of shattered beliefs. In three short months some people have to reassess some of their favorite standards. You might know some of them already. A few are; "All Chicago teams fold in the big game" (wasn't that them Bears in the Super Bowl?), "The weather can't get much worse" (was that before or after cold spell - thaw - rain - slush fall - freeze?), "Going into space is like a drive on the freeway" (enough said?), and lastly "You CAN teach an old new tricks" (I believe this one was uttered BEFORE 'Daddy Jack' announced that the 520's were headed to your local K - Mart).

So, by now all but the sleeping grizzilies have heard about Atari's plans to ship the 520ST off to the mass merchants and make Toys-R-Us his computer store. Right now there's a fly in the ointment. T-R-U, Sears, K-Mart, Target, Ect., Ect., Ex-ce-te-ra, just ain't listening to his little love song. They done gone an' made up their minds that the "home computer" revolution died some time back (Whoops, add another to the list!). Looks like all "DJ" has done is to go and make A LOT of dealers a tad angry. They don't view the 1040 as a significant upgrade of the current systems. Maybe, just maybe, if Atari would bring some of the product and enhancements to market that they have been talking about as recently as the March issue of BYTE the dealers and the undecided buyers would flock to Atari's products. As alluded to just now, the March issue of BYTE magazine has a cover story on the new 1040ST and an interview with Shiraz Shivji, Atari's VP or R&D. In it he talks about 2&4 MEG RAM machines, laser printers, expansion boxes w/IBM compatibility and other neat ideas. But alas, are we to be regaled with tales of wondrous delights and given more pre-sweetened Kool-Aid.
(Think anybody's listenin' Roy?)
(That's like asking if there's intelligent life out there, Gabby)

I DON'T KNOW, SOUNDS KINDA TRICKY TO ME

Want to upgrade your 800, XL or XE but aren't THAT brave? Well help is on the way in the form of Rice Electronics, 2956 Washington Blvd., Ogden, Utah 84401. These folks will sell you a 256K RAM upgrade for your 800XL computer for only $80. The kit includes 8-256K RAM chips, an assembled and tested board and color coded instructions. On top of that they will guarantee your computer to work if you follow the instructions and install the board correctly. You can send check or money order to the above address or use plastic and call them at 801-621-7423. In either case make sure to tell them you belong to Milatari and if ordering by mail send a copy of the newsletter and they say they will sent the group $5. Come on, you knew there was SOME reason for mentioning this didn't you?

Sometime soon we'll have to have a workshop on upgrading your computer and see how far we can take one.

SUCH A DEAL I'VE GOT .... (PART ???)

From the folks at Gumball Express (Honest people, I don't make 'em up) come two offers that are worth mentioning to our ST fans. The first is a combo offer featuring a Shanner (????) printer. The model SPC-700CI is a color printer that uses the Seikosha 4-hammer head system to print in seven basic colors (black, purple, red, magenta, green, cyan (blue for you commoners) and yellow) with a four color ribbon casette. While not the fastest at 50cps it does have multiple fonts and sizes and does graphics. It's an 80 col. printer with friction or pin feed and a Centronics parallel interface. The combo includes the printer, a driver program for the ST and two extra printer ribbons for only $279.95. Not bad.

The second offer is for the ST version of VIP Professional. This is a Lotus 1-2-3 look/work alike with some added features. VIP will handle up to 4MEG of memory (future ST's ??) and a spread sheet size of 8192 rows by 256 columns. It calculate to 300 digitsand has a mouse interface for ease of use or can emulate 123's text version if desired. Graphs and charts can be placed anywhere in the worksheet and can be moved around if wanted. The charts can also be exploded if you want them that way. It will handle multiple fonts and has 256 Query fields. The price for all this? Only $89.95. Now that is a good deal.
Where can you latch onto these goodies?
Glad you asked. (I just happen to have the address handy!)
Check or money order goes to;
Gumball Express
707 S.W. Washington St., Suite 200
Portland, OR 97205
or call and use card at, 1-800-423-6937. And be sure to mention that you're a member of Milatari.

Speaking of ST people, we will have some 3 1/2" disks at the next meeting. These are BASF SSDD and the price will be around $1.65 ea..

YES, YOUR VOTE CAN MAKE A DIFFERENCE
(JUST ASK MR. MARCOS)

Elections are just around the corner and the nomination committee is forming. If you'd like to help select a slate of candidates to run for office, give Dave a call or see him at the next meeting. He will be glad to let you help.

We will need almost a full slate of candidates as treasurer Steve Tupper's two terms are up and Dave has decided not to run for re-election (right Dave?) Personally I think this Marcos thing shook him up a little. So if any of you would like to try your hand at running Milatari from one office or another now is the time to throw your hat into the ring. Make sure you take it off first and don't let it hit the "Hulkster".
The answer to THAT question is NO.
You were going to ask it weren't you?

SAY BYE...

Would have more to say but both Dave and Roy took their phones off the hook when I tried to call and dump this to them. So we had to cut it short.
See you on the 15th.....

ser-defined object could be used in this tree, with the result that your application's code would be entered from the AES whenever the desktop was redrawn. This would allow you to put VDI pictures or complex images onto the desktop background.

A SIN OF OMISSION

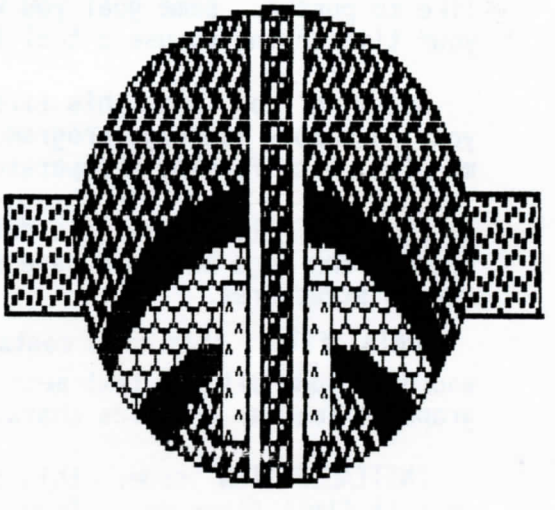
In the last column, I neglected to mention that strings whose addresses are passed in the WF_NAME and WF_INFO wind_set calls must be allocated in a static data area. Since the AES remembers the addresses (not the characters), a disaster may result if the storage has been reused when the window manager next attempts to draw the window title area.

COMING SOON...

This concludes our tour of GEM's basic window management techniques. There have been some unavoidable glimpses of paths not yet taken (forward references), but we will return in time.

On our next excursion, we will take a look at techniques for handling simple dialog boxes, and start exploring the mysteries of resources and object trees.

```
>>>>>>>>>>>>>>>>>>>>>>>>>>> Sample Redraw Code <<<<<<<<<<<<<<<<<<<<<<<<<<<<

        VOID
do_redraw(wh, area)             /* wh = window handle from msg[3] */
        WORD    wh;             /* area = pointer to redraw rect- */
        GRECT   *area;          /*   tangle in msg[4] thru msg[7] */
        {
        GRECT   box;

        graf_mouse(M_OFF, 0x0L);
        wind_update(BEG_UPDATE);

        wind_get(wh, WF_FIRSTXYWH, &box.g_x, &box.g_y, &box.g_w, &box.g_h);
        while ( box.g_w && box.g_h )
                {
                if (rc_intersect(full, &box))      /* Full is entire screen */
                if (rc_intersect(area, &box))
                        {
                        if (wh == w1_handle)       /* Test for window 1 handle */
                                {                  /* AES redraw example       */
                                objc_draw(w1_tree, ROOT, MAX_DEPTH, box.g_x,
                                        box.g_y, box.g_w, box.g_h);
                                }
                        else if (wh == w2_handle) /* Test for window 2 handle */
                                {                  /* VDI redraw example       */
                                set_clip(TRUE, &box);
                                /*  Put VDI drawing calls here */
                                }
                        /* add more windows here */
                        }
                wind_get(wh, WF_NEXTXYWH, &box.g_x, &box.g_y, &box.g_w,
                        &box.g_h);
                }

        wind_update(END_UPDATE);
        graf_mouse(M_ON, 0x0L);
        }

>>>>>>>>>>>>>>>>>>>>>>>>>>> Utilities used in do_redraw <<<<<<<<<<<<<<<<<<<<<<<<<<<

        VOID
set_clip(clip_flag, area)       /* set clip to specified area   */
        WORD    clip_flag;
        GRECT   *area;
        {
        WORD    pxy[4];

        grect_to_array(area, pxy);
        vs_clip(vdi_handle, clip_flag, pxy);
        }

        VOID
grect_to_array(area, array)     /* convert x,y,w,h to upr lt x,y and    */
        GRECT   *area;          /*                  lwr rt x,y          */
        WORD    *array;
        {
        *array++ = area->g_x;
        *array++ = area->g_y;
        *array++ = area->g_x + area->g_w - 1;
        *array = area->g_y + area->g_h - 1;
        }

        WORD
        WORD    tx, ty, tw, th;

        tw = min(p2->g_x + p2->g_w, p1->g_x + p1->g_w);
        th = min(p2->g_y + p2->g_h, p1->g_y + p1->g_h);
        tx = max(p2->g_x, p1->g_x);
        ty = max(p2->g_y, p1->g_y);
        p2->g_x = tx;
        p2->g_y = ty;
        p2->g_w = tw - tx;
        p2->g_h = th - ty;
        return( (tw > tx) && (th > ty) );
        }

>>>>>>>>>>>>>>>>>>>>>>>>>>> "Self-redraw" Utility <<<<<<<<<<<<<<<<<<<<<<<<<<<

        VOID
send_redraw(wh, p)
        WORD    wh;
        GRECT   *p;
        {
        WORD    msg[8];

        msg[0] = WM_REDRAW;             /* Defined in GEMBIND.H     */
        msg[1] = gl_apid;               /* As returned by appl_init */
        msg[2] = 0;
        msg[3] = wh;                    /* Handle of window to redraw */
        msg[4] = p->g_x;
        msg[5] = p->g_y;
        msg[6] = p->g_w;
        msg[7] = p->g_h;
        appl_write(gl_apid, 16, &msg);  /* Use ADDR(msg) for portability */
        }

>>>>>>>>>>>>>>>>>>>>>>>>>>> Utilities for Window Requests <<<<<<<<<<<<<<<<<<<<<<<

        VOID
rc_constrain(pc, pt)
        GRECT           *pc;
        GRECT           *pt;
        {
        if (pt->g_x < pc->g_x)
                pt->g_x = pc->g_x;
        if (pt->g_y < pc->g_y)
                pt->g_y = pc->g_y;
        if ((pt->g_x + pt->g_w) > (pc->g_x + pc->g_w))
                pt->g_x = (pc->g_x + pc->g_w) - pt->g_w;
        if ((pt->g_y + pt->g_h) > (pc->g_y + pc->g_h))
                pt->g_y = (pc->g_y + pc->g_h) - pt->g_h;
        }

        WORD
align(x,n)              /* Snap position x to an n-bit grid          */
        WORD    x, n;   /* Use n = 16 for horizontal word alignment */
        {
        x += (n >> 2) - 1;              /* Round and... */
        x = n * (x / n);                /* remove residue */
        return (x);
        }

>>>>>>>>>>>>>>>>>>>>>>>>>>> Window full utility <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        VOID
hndl_full(wh)           /* depending on current window state, make window    */
        WORD    wh;     /*   full size -or- return to previous shrunken size */
        {               /* graf_ calls are optional special effects.         */
        GRECT   prev;
        GRECT   curr;
        GRECT   full;

        wind_get(wh, WF_CXYWH, &curr.g_x, &curr.g_y, &curr.g_w, &curr.g_h);
        wind_get(wh, WF_PXYWH, &prev.g_x, &prev.g_y, &prev.g_w, &prev.g_h);
        wind_get(wh, WF_FXYWH, &full.g_x, &full.g_y, &full.g_w, &full.g_h);
        if ( rc_equal(&curr, &full) )
                {               /* Is full, change to previous          */
                graf_shrinkbox(prev.g_x, prev.g_y, prev.g_w, prev.g_h,
                        full.g_x, full.g_y, full.g_w, full.g_h);
                wind_set(wh, WF_CXYWH, prev.g_x, prev.g_y, prev.g_w, prev.g_h);
                        /* put send_redraw here if you need it */
                }
        else
                {               /* is not full, so set to full          */
                graf_growbox(curr.g_x, curr.g_y, curr.g_w, curr.g_h,
                        full.g_x, full.g_y, full.g_w, full.g_h);
                wind_set(wh, WF_CXYWH, full.g_x, full.g_y, full.g_w, full.g_h);
                }
        }

        WORD                    /* tests for two rectangles equal       */
rc_equal(p1, p2)
        GRECT   *p1, *p2;
        {
        if ((p1->g_x != p2->g_x) ||
            (p1->g_y != p2->g_y) ||
            (p1->g_w != p2->g_w) ||
            (p1->g_h != p2->g_h))
                return(FALSE);
        return(TRUE);
        }
```



*** CHRIS CRAWFORD *** ASSEMBLY LANGUAGE COURSE

LESSON EIGHT: SOME ADVANCED TOPICS

We have covered all of the traditional material associated with 6502 assembly language programming. However, there remain a number of topics that should be addressed before we finish. They are not closely associated with each other, so I will take them in random order.

The first topic is perhaps the most difficult one for a beginning assembly-language programmer: Where do I begin? How do I put together an entire assembly language project?

The problem here is seldom a technical one. Most beginners are stopped by their own lack of goals rather than any lack of technical expertise. One does not just write an assembly language program because one knows assembly language -- that is putting the cart before the horse. One starts with goals and then considers means.

A story from my early days with micros will illustrate this point. I did not have anybody to teach me assembly language. I decided in 1976 that I wanted to do wargames on computers. Accordingly, I bought a KIM-1, an early 6502-based single-board computer. I received it in January 1977. I studied the manuals and taught myself 6502 machine language. I had my first wargame up and running in six weeks. That means that I not only taught myself 6502 in six weeks, but I wrote and debugged a program at the same time.

Now, the point of this story is NOT "Wow, isn't Chris Crawford the smartest programmer who ever lived!" The point of this story is that goal-oriented learning is far more effective than goal-less learning. Had I sat in on some technical course on 6502, I would have taken months and months to learn the material. Because I had a clear goal, I learned very quickly.

My advice to you, the beginning assembly programmer, is this: You have acquainted yourself with the rudiments of 6502 programming. If you have some project you would like to pursue, some goal you would like to achieve, then do it. If not, don't waste your time trying to use a tool for its own sake.

Assuming you pass this first test, there remains the broad problem of organizng your assembly language program. I suggest that you break your program up into six modules, each forming a separate source code file. These six modules would be:

EQUATES file: this file defines all of the equates used by the program: the data areas, the page zero and page six usage, and perhaps some of the large graphics and screen structures.

DATA file: this file contains all of the static tables used by the program. This would include all the text messages that would be printed onto the screen, bitmaps of graphics images, graphics character set definitions, and so forth.

INITIALIZATION code: this file contains the routines that initialize the program when it first fires up. They set up the screen, clear out all the special graphics

and sound registers, zero out all the arrays that need to be cleared, and do all the other legwork associated with clearing the decks for a program.

INTERRUPT code: this module contains the code associated with any interrupts used by your program. This would most commonly involve vertical blank interrupts and display list interrupts. Inasmuch as your interrupts should be well-separated from your other code, you might as well keep the code in a separate file.

MAINLINE code. This includes the main program loop that controls the primary behavior of the program. If you have problems imagining this, think of it as nothing more than a series of subroutine calls arranged in a loop, with each subroutine handling one chunk of the overall process.

SUBROUTINE code: After a while you build up a collection of subroutines for handling standard processes in the program. Keep them here.

The second topic I would like to talk about is the place of the 6502 in the larger world of microprocessors. The 6502 is undoubtedly the most successful of microprocessors to date, having been installed in more systems than any other microprocessor. It is also a very old microprocessor, having first appeared in 1976. That makes it nine years old.

A very simple way to approach the world of microprocessors is to group them into two sets -- the Sixes and the Eights. The Eights represent the earliest group of microprocessors, they trace their lineage al the way back to the 4004, the first microprocessor. The 4004 was followed by the 8008, the first eight-bit micorprocessor. The 8008 was superseded by the 8080, which was in turn followed by the Z-80. The Z-80 was the most advanced eight-bit processor in the Eights line. The next step was to go to 16 bits with the 8088 and 8086. These were followed by the more powerful 80186, 80286, and 80386.

The fundamental philosophy of all the Eights can be expressed in two words: features and compatibility. The designers of the Eights were always adding new features to the microprocessors with each successive generation. The goal seemed to be to pack as many bells and whistles in as would fit. The second goal, compatibility, meant compatibility with the previous microprocessor in the series. This insured that software developed for previous versions would still run on the newer versions.

The result of this design philosophy was a series of powerful microprocessors that were quite complex in layout and rather difficult to learn. The features were piled up on each other in a bewildering array. Once you learn the system, it seems natural enough. But it is something of a mess.

The Sixes include the 6800, the 6502, the 6809, and the 68000. The two key words guiding the design of the Sixes are cleanliness and speed. The idea was to make the instruction sets clean, powerful, and fast. The hope was that the processors would be so easy to learn that compatibility would not be a problem. The design approach was to use just a few simple instructions, but give them variations that greatly extend their power. Thus, the 6502 has a LDA instruction that can be used with a great many addressing modes.

The 68000 is the 32 bit entry into the Sixes line. It carries the idea of cleanliness even further than the 6502. The 68000 uses a single instruction with different modes to replace the 6502 instructions LDA, LDX, LDY, STA, STX, STY, TXA,

TAX, TYA, PHA and PLA. That's quite a simplification!

The 68000 also boasts sixteen registers, each 32 bits wide. That's a total of 512 bits of register space, the 6502 has 32 bits of equivalent register space. Those sixteen registers eliminate many of the data-shuffling problems so common with the 6502.

The 68000 has a linear address space 24 bits wide -- that's sixteen megabytes! Thus, a 68000 can directly address 16 megabytes of RAM and ROM. The 6502, by contrast, can only address 64K directly -- it must use paging sytems that slow it down to address more memory.

Finally, the 68000 has a number of advanced capabilities that make possible a number of special capabilities. I will describe just one stack frames. The 68000 makes it easy to set up a local, temporary stack when you enter a subroutine. Thus, subroutines can have their own local variables stored on the stack, accessed via a special stack pointer register. The 68000 will manage all the housekeeping necessary to keep such a system straight.

Did I mention that 68000 has hardware multiply/divide?

****** THE END ******

PERSONAL PASCAL BY OSS
Reviewed by Roy Duvall

This product is a complete programming environment, as well as, an extended ISO pascal compiler. The manual contains a very descriptive and informative discussion of the features, functions and interfaces to GEM, VDI, AES, TOS. The manual does not teach pascal and its examples are only program fragments. But it does cover most of the routines and syntax requirements of pascal.

The package includes a standard, full featured screen editor, the pascal compiler and include files, a standard linker and its associated libraries. The ability to use assembler or C language routines is also available through the use of the standard linker.

While I'm not a novice to programming I certainly had great difficulty writing any working programs before I had Personal Pascal. Due to the lack of information about GEM / TOS and the convenience of a programming environment. Personal Pascal certainly addressed both of these problems.

After reading through the documentation I finally realized the beauty and power of GEM / TOS. In my first attempt I successfully wrote and compiled a working program. For me that is simply amazing! This is by far the BEST programming environment available for the 520ST today! The high standards of Quality and ease of use, always found in an OSS product, is at its best here. I did run into some problems and called the OSS Bulletin Board (408-446-3451) to see if I could find an answer. I left a message describing my problem and to my great surprise I received a phone call with my answer a few days later. I recently call the OSS board again and found many new additional documentation files and sample programs. I find that this is a wonderful benefit for those who can afford a long distance phone bill.

So far I have discovered no major flaws in this system. I Highly recommend PERSONAL PASCAL to anyone seriously interested in writing programs for the 520ST. The documentation is outstanding and the support is even better!

MILWAUKEE AREA ATARI USER'S GROUP AND NEWSLETTER INFORMATION

MILATARI OFFICERS

President	David Frazer	542-7242
Vice President	Carl Mielcarek	355-3539
Secretary		
Treasurer	Steve Tupper	462-8178

MILATARI VOLUNTEERS

MILATARI West	Dennis Bogie	968-2361
Education SIG	Joe Sanders	447-1660
ATR8000 & CP/M		
Adv. Lang. SIG	Erik Hanson	252-3146
Database	Ron Friedel	354-1717
BBS Sysop	Richard Dankert	781-2338
Membership	Dave Bogie	968-2361
Newsletter	Roy Duvall	363-8231
Public Domain Libraries		
Cassette	Lee Musial	466-9140
Disk	Dennis Wilson	476-7767
	Bill Lawrence	968-3082
Copyright Library		
Cassette/Disk		
Publications	Bill Feest	321-4314
	Milatari BBS	
	414-781-5710	

NEWSLETTER INFORMATION

This newsletter is written and printed by members of the Milwaukee Area ATARI User's Group (MILATARI). Opinions expressed in this publication are those of the individual authors and do not necessarily represent or reflect the opinions of the Milwaukee Area Atari User's Group, its officers, its members or its advertisers except where noted. Articles reprinted from bulletin boards and other newsletters are presented as an information exchange, no warranty is made to their accuracy or factuality.

Your contributions of articles are always welcome. You may submit your article on ATARI compatible cassette or diskette, on typewritten form or you can arrange with the editor to upload your file via modem. You can send Graphics eight or seven plus screens stored on disk in Micropainter or Micro Illustrator formats.

Milwaukee Area Atari User's Group

MILATARI is an independent, user education group which is not affiliated with ATARI INC. The newsletter is the official publication of MILATARI and is intended for the education of its members as well as for the dissemination of information concerning ATARI computer products.

MILATARI membership is open to individuals and families who are interested in using and programming ATARI computers. The membership includes a subscription to this newsletter and access to the club libraries. The annual membership fee is $15 for individuals or $20 for a family.

Vendors wishing to display and/or sell items at MILATARI meetings must make prior arrangements with the club vice president. Rates are $10 per meeting or $90 per year payable in advance.

Other computer user groups may obtain copies of this newsletter on an exchange basis.

MILATARI ADVERTISING RATES

This newsletter will accept camera ready advertising copy from anyone supplying goods and services of interest to our membership.

Current paid members of MILATARI may place classified ads in the newsletter at no charge.

Advertising Rates

Full page	$37.50
Half page	$20.00
Quarter page	$12.50
Business card	$2.00

MILWAUKEE AREA ATARI USERS CLUB

U.S. POSTAGE PAID
BULK RATE
PERMIT NO. 201
WAUKESHA, WI 53187

MILWAUKEE AREA ATARI USERS GROUP
Post Office Box 19858
West Allis, Wisconsin 53219-0858

Address Correction Requested
Forwarding Postage Guaranteed